DO NOWEJ PODSTAWY PROGRAMOWEJ

Część 1

Inżynieria programowania – projektowanie oprogramowania, testowanie i dokumentowanie aplikacji

Kwalifikacja INF.04

Projektowanie, programowanie i testowanie aplikacji

Podręcznik do nauki zawodu
technik programista

Angelika Krupa, Weronika Kortas

Helion

Redaktor prowadzący: Joanna Zaręba

Projekt okładki: Jan Paluch

Ilustracja na okładce została wykorzystana za zgodą Shutterstock.

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie?inf041
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-283-7415-7

Printed in Poland.

Spis treści

1

Przygotowanie środowiska pracy

Przygotowanie środowiska pracy to nie tylko instalacja i konfiguracja niezbędnych aplikacji i narzędzi, ale również nauka optymalnego ich wykorzystania. Obecnie wszelkiego rodzaju tutoriale i szkolenia e-learningowe niemal „wylewają się" z internetu, dlatego coraz częściej wymagana jest znajomość wielu narzędzi już na początku pracy — po to by proces tworzenia oprogramowania był jak najszybszy i jak najbardziej wydajny.

1.1. Narzędzia wykorzystywane w prowadzeniu projektu

W dzisiejszych czasach realizacja projektów jest tak różnorodna, że rynek nieustannie dostarcza nowych rozwiązań technologicznych, które mają za zadanie wspomagać procesy i zarządzanie projektami. W niniejszym rozdziale zostaną omówione narzędzia (zarówno płatne, jak i darmowe), które wspierają zarządzanie projektem, zarządzanie testami oraz wytwarzanie oprogramowania.

UWAGA

Rozdział ma charakter wprowadzający — warto mieć na uwadze, że niżej wymienione narzędzia cały czas są rozwijane i wzbogacane o kolejne funkcje.

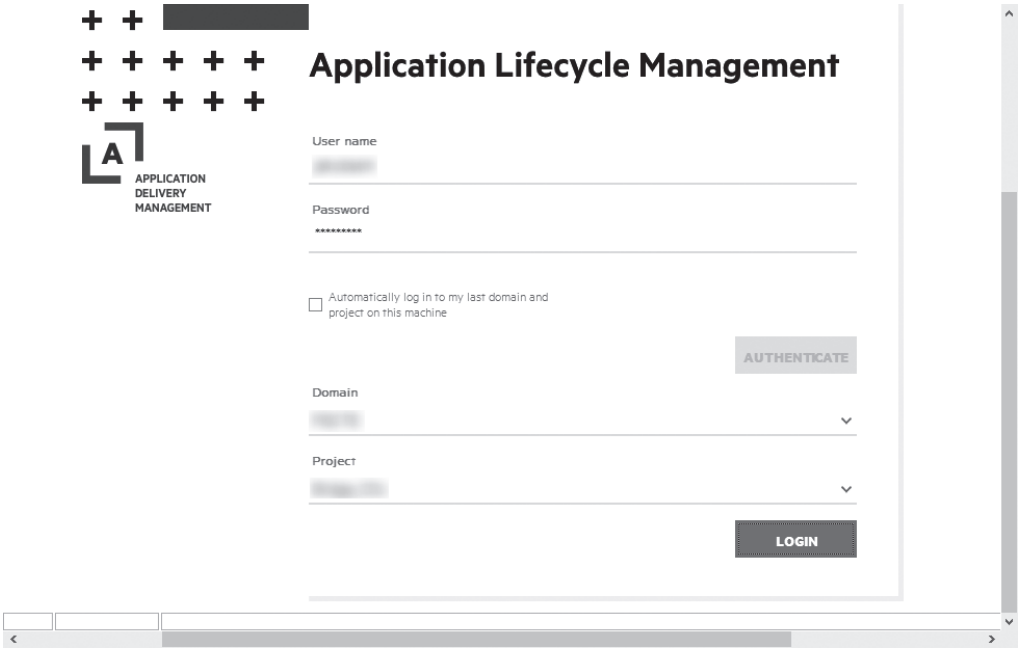
1.1.1. HP ALM

Jako pierwsze zostanie omówione narzędzie HP ALM. Jest to aplikacja do wspierania procesu testowego i zarządzania testami. Dostęp do narzędzia jest przyznawany na podstawie licencji (1 licencja = 1 użytkownik). Umożliwia ono planowanie oraz wykonanie testów manualnych i automatycznych, zarządzanie defektami (zgłaszanie, monitorowanie). HP ALM dzieli się na dwie części:

- administracyjną (ang. *Site Administration*);
- użytkownika (ang. *ALM Platform*) — cały moduł przeznaczony dla użytkowników.

W dalszej części rozdziału skupimy się na części użytkownika, gdyż panel administracyjny jest dostępny — jak sama nazwa wskazuje — dla osób pełniących funkcję administratora. W panelu administracyjnym wykonuje się wszystkie czynności administracyjne, takie jak tworzenie domen, projektów, użytkowników oraz nadawanie uprawnień.

Po wybraniu adresu docelowego narzędzia w danej organizacji pojawia się ekran logowania (rysunek 1.1):



Rysunek 1.1. Ekran logowania w HP ALM

Jak można zauważyć, poza wymaganymi danymi do logowania aplikacja wymusza na nas wybór z list rozwijanych domeny (ang. *Domain*) i projektu. Użytkownik może się zalogować wyłącznie do projektów, do których nadano mu dostęp.

UWAGA

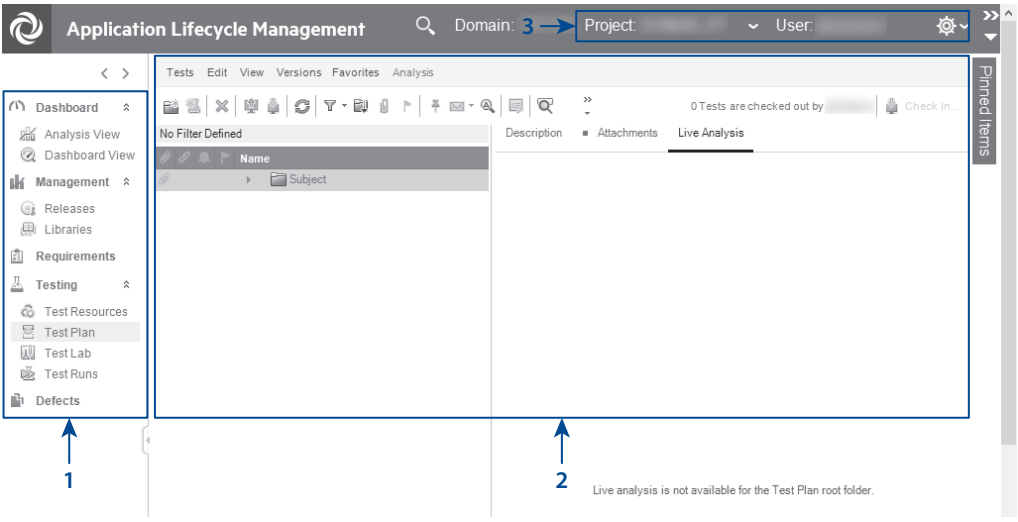
Po zalogowaniu widoczne są tylko te funkcje, do których mamy dostęp.

Panel użytkownika może się składać z następujących modułów (widoczność zależna od uprawnień):

- *Dashboard* — statystyki, raporty;
- *Management* — zarządzanie planem wersji;
- *Requirements* — definiowanie wymagań systemowych i zarządzanie nimi;
- *Testing* — tworzenie zestawów testów w projektach, uruchamianie zaplanowanych testów;
- *Defects* — tworzenie i śledzenie statusów defektów.

Poniżej przedstawiono podgląd widoku ekranu po zalogowaniu do aplikacji (rysunek 1.2). Widać na nim trzy grupy menu:

1. Menu wyboru modułów.
2. Menu operacji dostępnych w wybranym wcześniej module.
3. Menu użytkownika (wybór projektu, ustawienia konta).



Rysunek 1.2. Widok na menu i moduły w HP ALM

W dalszej części podrozdziału przedstawione zostaną moduły *Testing* i *Defects*.

1.1.1.1. Moduł Testing

Jest to moduł służący do organizowania procesu testowego. Składa się z następujących elementów (użytkownik widzi tylko te, do których ma nadany dostęp):

- *Test Resources* — zawiera wszystkie dodatkowe zasoby używane do testów manualnych oraz automatycznych, zarówno funkcjonalnych, jak i wydajnościowych.
- *Business Components* — moduł, który pozwala zarządzać komponentami biznesowymi.
- *Test Lab* — tu przenosimy testy, które są skonfigurowane i gotowe do uruchomienia.
- *Test Runs* — w module widoczne są wszystkie uruchomienia testów wraz z ich wynikami.
- *Test Plan* — moduł pozwalający na zorganizowanie całego procesu testowania poprzez określenie, jakie testy zostaną wykonane w celu zweryfikowania poprawności działania systemu.

DEFINICJA

Weryfikacja — egzaminowanie poprawności i dostarczenie obiektywnego dowodu na to, że produkt procesu wytwarzania oprogramowania spełnia zdefiniowane wymagania.

Walidacja — sprawdzenie poprawności i dostarczenie obiektywnego dowodu na to, że produkt procesu wytwarzania oprogramowania zaspokaja potrzeby użytkownika i spełnia jego wymagania.

1.1.1.2. Moduł Defects

Jest to moduł do zarządzania błędami, który daje możliwość ich rejestrowania, przypisywania do konkretnych osób i zmiany statusów zarejestrowanych błędów.

Możemy zgłosić tutaj wszystkie uwagi do zachowań (zarówno funkcjonalnych, jak i niefunkcjonalnych) systemu odbiegających od przyjętych wymagań.

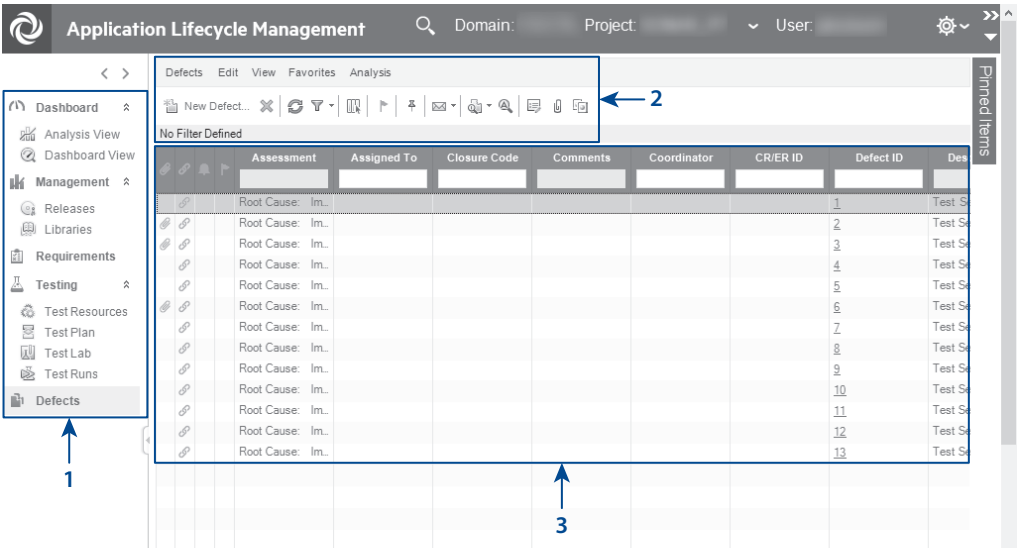
DEFINICJA

Wymaganie funkcjonalne dotyczy wyniku zachowania systemu, który powinien zostać dostarczony przez funkcję systemu.

Wymaganie niefunkcjonalne dotyczy jakości tworzonego oprogramowania, ale nie zostało określone przez wymagania funkcjonalne. Odnosi się m.in. do wydajności, dostępności, niezawodności, skalowalności i przenośności.

Poniżej przedstawiono podgląd widoku ekranu modułu *Defects* (rysunek 1.3). Można na nim zauważyć trzy grupy menu:

1. Menu wyboru modułów.
2. Menu górne.
3. Widok na wszystkie zgłoszenia w ramach projektu, do którego jesteśmy zalogowani.



Rysunek 1.3. Widok narzędzia w module Defects

Defekty możemy zgłosić zarówno z poziomu modułu *Test Lab*, jak i z poziomu *Defects*. W menu górnym (2) należy najpierw wybrać *New Defect*, a następnie wypełnić wszystkie wymagane pola w formularzu. Po dodaniu defektu jest on widoczny na liście (3).

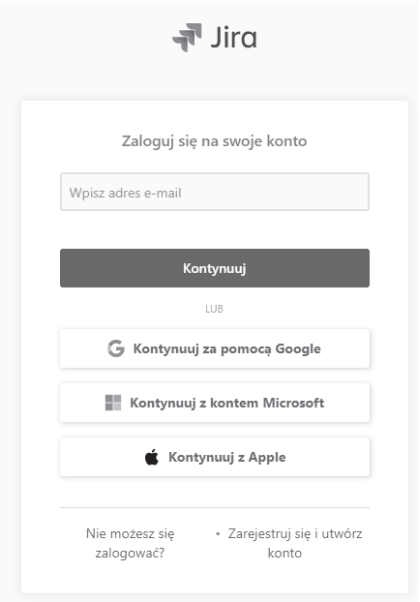
HP ALM jest bardzo elastycznym narzędziem. Możemy je spersonalizować do każdego projektu. Co ważne — możemy sami definiować raporty czy widoki i udostępniać je innym osobom.

Jest to jedno z najdroższych narzędzi na rynku, jednak jest warte swojej ceny.

1.1.2. Jira

Jira jest komercyjnym narzędziem nie tylko wykorzystywanym w zarządzaniu projektami, ale również wspomagającym elektroniczny obieg dokumentacji w firmie.

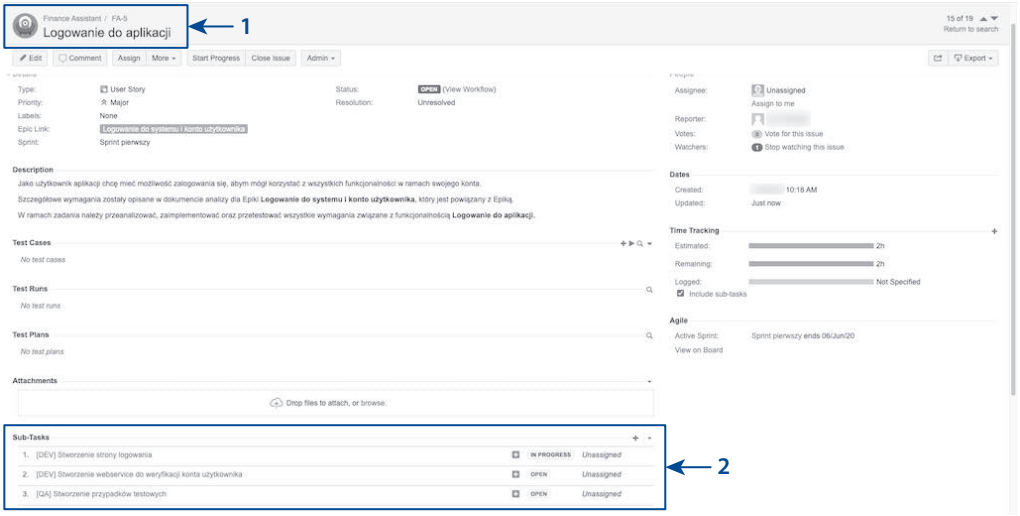
Tak wygląda ekran logowania dla Jira cloud (rysunek 1.4):



Rysunek 1.4. Ekran logowania w narzędziu Jira

Warto zaznaczyć, że ekran logowania do wersji cloud różni się wizualnie od wersji serwerowej. Jira dostosowuje się do projektu. Zależnie od metodyki, w jakiej projekt jest realizowany, narzędzie daje nam całą paletę różnorodnych funkcji. Poniżej (rysunek 1.5) zaprezentowano widok przykładowego zadania w systemie Jira wraz z podzadaniami (ang. *subtasks*). Można na nim zauważyć, że:

1. zadanie główne wyświetlane jest z nazwą projektu;
2. widoczne są wszystkie podzadania powiązane z zadaniem głównym.



Rysunek 1.5. Widok projektu w systemie Jira

W momencie gdy wszystkie podzadania wejdą w status świadczący o zakończonych pracach (może to być *Resolved* lub *Closed*), uznaje się, że zadanie główne może być wdrażane do środowiska (docelowego) produkcyjnego.

DEFINICJA

Środowisko produkcyjne — zainstalowane w siedzibie klienta, firmy tworzącej oprogramowanie lub w chmurze sprzęt i oprogramowanie, w których moduł (lub system) będzie używany. W skład oprogramowania mogą wchodzić systemy operacyjne, bazy danych i inne aplikacje.

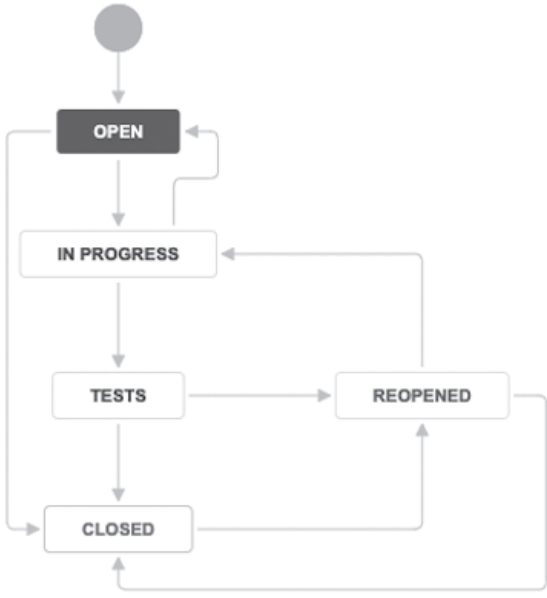
Dodatkowym atutem narzędzia Jira jest definiowanie cyklu życia (ang. *workflow*) błędu.

DEFINICJA

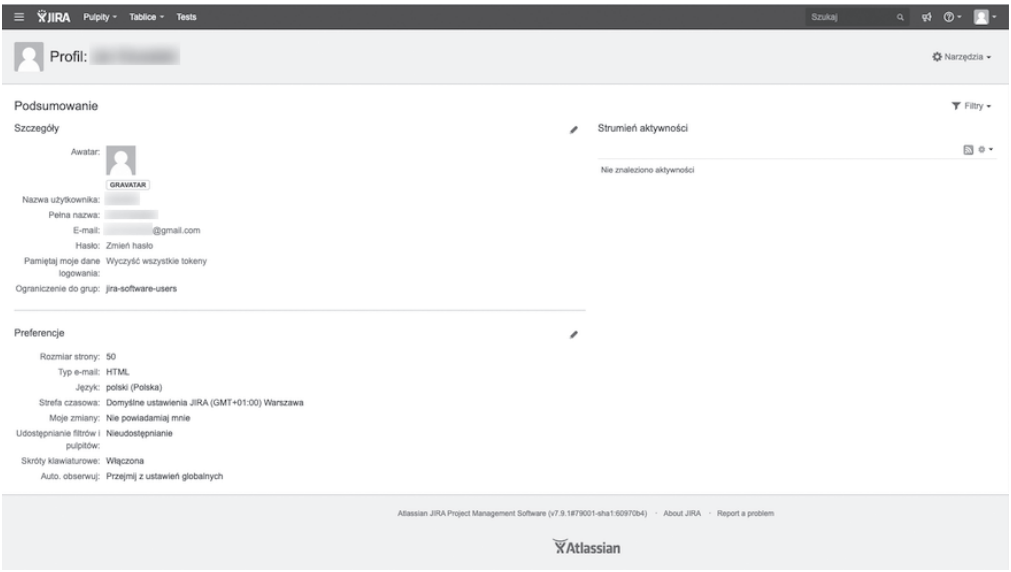
Cykl życia błędu to proces, przez który przechodzi usterka oprogramowania. Rozpoczyna się, gdy błąd zostanie wykryty, a kończy w chwili jego usunięcia, po wcześniejszym upewnieniu się, że nie został zduplikowany.

Na rysunku 1.6 przedstawiony jest diagram, który możemy dowolnie modyfikować. Każdy projekt może mieć odrębny *workflow*.

Rysunek 1.6.
Przykładowy cykl życia błędu (ang. workflow)



W *Panelu użytkownika* (rysunek 1.7) możemy zmienić preferencje co do wyświetlania stron oraz sprawdzić historię aktywności.



Rysunek 1.7. Konto użytkownika

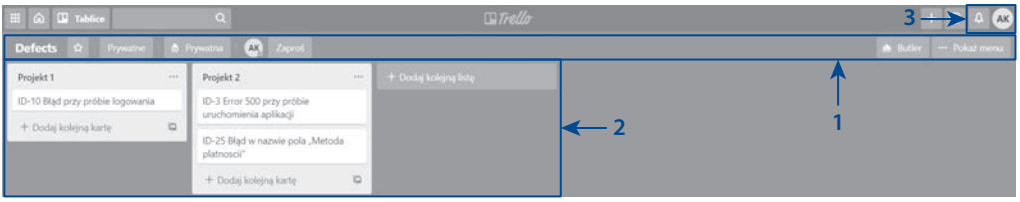
Dodatkowe informacje o narzędziu Jira zostaną podane w rozdziale 4.

1.1.3. Trello

Trello jest narzędziem bezpłatnym, które może być wykorzystywane zarówno w projektach komercyjnych, jak i w celach prywatnych (np. do planowania podróży, budżetu domowego, remontu).

Poniżej przedstawiono podgląd na widok tablicy Trello (rysunek 1.8). Można na nim zauważyć trzy grupy menu:

1. Menu operacji dotyczących udostępniania/prywatności tablicy.
2. Widok kart.
3. Konto użytkownika wraz z opcją powiadomień.



Rysunek 1.8. Widok na tablicę Trello

Dodatkowe informacje o Trello zostaną zaprezentowane w rozdziale 4.

W dalszej części podręcznika zostanie omówiony jeszcze TestLink — darmowe narzędzie typu *open source*.

1.1.4. Dobór narzędzi do zarządzania projektem

Poniżej przedstawiono krótkie zestawienie porównawcze narzędzi, które ułatwi Ci dopasowanie narzędzi do projektu. Zostały w nim uwzględnione ich kluczowe funkcje, dzięki którym wspierają prowadzenie projektu.

Tabela 1.1. Zestawienie porównawcze narzędzi

	HP ALM	Jira	Trello	TestLink
Płatna licencja	✓	✓		
Możliwość rozszerzania funkcji dzięki płatnym dodatkom	✓	✓	✓	
Personalizacja raportów	✓	✓		
Definiowanie cyklu życia błędu	✓	✓		
Wykonywanie przypadków testowych	✓			✓
Obsługa testów automatycznych	✓	✓		✓

1.2. Narzędzia wykorzystywane w wytwarzaniu oprogramowania

Jednym z elementów wytwarzania oprogramowania jest tworzenie aplikacji za pomocą wybranego języka programowania. W tym podręczniku skupiamy się na języku JavaScript. Zacznijmy od opisu elementów, na jakie należy zwrócić szczególną uwagę podczas instalacji oprogramowania.

Narzędzia pomocne do napisania aplikacji przeglądarkowej to Git i WebStorm.

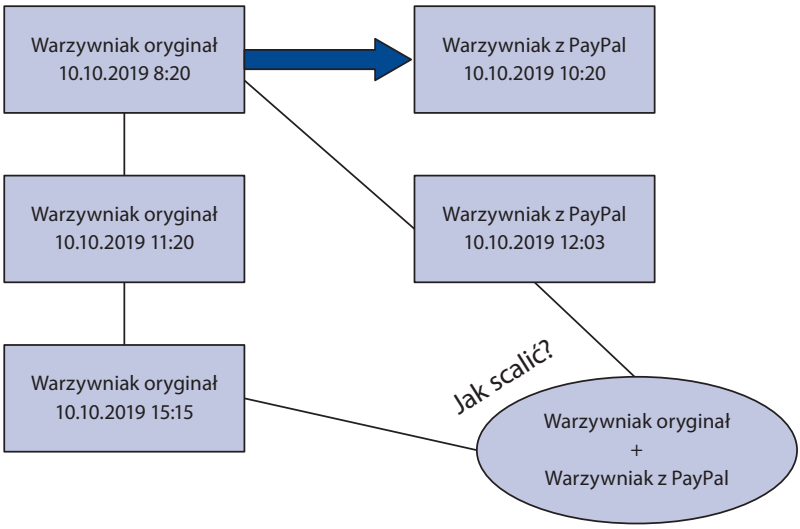
1.2.1. Instalacja programów WebStorm i Git

1.2.1.1. Git — system kontroli wersji

Wyobraźmy sobie wspólne pisanie fragmentów kodu przez wielu użytkowników i przenoszenie tych kodów za każdym razem do chmury. Jak zapewnić, by zmiany dodawane przez dowolnego programistę były widoczne dla pozostałych współpracowników?

Gdy kolega z zespołu pobierze plik, a następnie coś w nim zmieni, wówczas, by było wiadomo, że kod został zmieniony, można dodać katalog z datą i godziną zmiany i to w nim właśnie przechowywana będzie nowa wersja pliku, podczas gdy oryginalna pozostanie niezmieniona.

Załóżmy ponadto, że kolega pracuje nad nową funkcją. Skopiował więc cały katalog oryginalny i nanosi zmiany wewnątrz kopii folderu. Poprawki do bieżącej funkcji wciąż są natomiast rozwijane w oryginalnym folderze (rysunek 1.9).



Rysunek 1.9. Symulacja prowadzenia projektu zapisanego w folderach w chmurze

Pewnego dnia kolega skończył pisanie funkcji, nad którą pracował. Trzeba więc połączyć dwa różne katalogi. Gdyby nie istniały systemy kontroli wersji, trzeba byłoby porównać wszystkie pliki i nanosić ręcznie niezbędne zmiany. Przy dużej liczbie plików i zmian byłby to spory problem.

Na szczęście problem ten pozwalają łatwo rozwiązać wspomniane już narzędzia do kontroli wersji, a wśród nich to bodaj najpopularniejsze — Git.

DEFINICJA

Git to repozytorium kodu, czyli w uproszczeniu narzędzie do przenoszenia plików między komputerem, na którym jest tworzony kod, a innym serwerem, na którym przechowywana jest kopia. Zapewnia wersjonowanie plików (dlatego nazywane jest też systemem kontroli wersji), czyli przechowywanie w osobnych gałęziach (ang. *branch*), różnych wersji kodu pochodzących od jednej głównej wersji, nazywanej gałęzią główną (ang. *master*). Każda gałąź może być rozwijana niezależnie, a potem scalana do jednolitej wersji.

Przykład 1.1

Zobaczmy na przykładzie, jak rozwiązać problemy z integracją.

Od czasu do czasu w lokalnej społeczności organizowane są wyprzedaże garażowe. Załóżmy, że rodzina Nowaków chciałaby się przyłączyć do tej wyprzedaży i opróżnić już pokoje dzieci — Antka i Marysi — bo zalegają w nich pluszaki i ubrania, z których dzieci już wyrosły.

W pokoju Antka odnaleziono m.in. garnitur komunijny, pluszowego koalę oraz o trzy rozmiary za małe buty.

W pokoju Marysi odnaleziono za małe biurko, za niskie krzesło obrotowe oraz mnóstwo szpargałów z wakacji w Egipcie.

Antek zdecydował, że oddaje wszystkie rzeczy wstępnie zakwalifikowane do wystawienia na wyprzedaż. Przeniósł je do salonu, gdzie były składowane przedmioty do sprzedania.

Marysia nie mogła pożegnać się z pamiątkami z Egiptu. „Oddaję biurko i krzesło, ale pamiątki zostają" — orzekła.

Załóżmy, że organizację wyprzedaży przeprowadzamy za pomocą Gita. Najpierw musimy wybrać rzeczy na wyprzedaż. Robimy to za pomocą polecenia w postaci `git add nazwa_rzeczy`.

Wybranie rzeczy (przeniesienie ich do salonu) za pomocą Gita wyglądałoby więc tak:

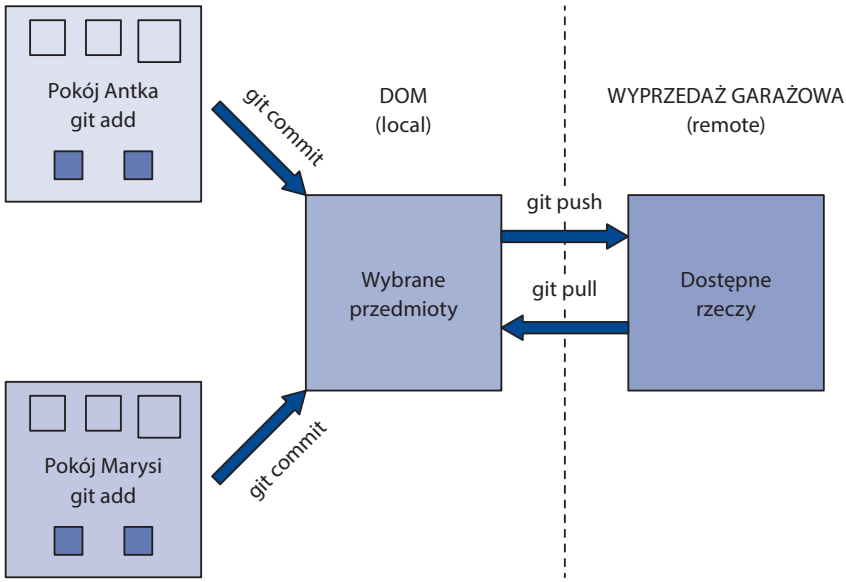
```
git add garnitur
git add pluszak
git add buty
git add biurko
git add krzeslo
```

Gdy wszystkie przedmioty zostały już wybrane (czyli przeniesione do salonu), podjęta została ostateczna decyzja zatwierdzająca wybór. Realizujemy ją za pomocą polecenia `git commit -m 'rzeczy na wyprzedaz'` (`-m` to parametr, który oznacza komentarz do commita, czyli wprowadzonych zmian). Nie znaczy to jednak, że rzeczy zostały już przeniesione do miejsca, gdzie będą sprzedawane, czyli do garażu sąsiada (to będzie nasze repozytorium zdalne — `remote`). Wciąż jeszcze znajdują się w salonie (czyli repozytorium lokalnym — `local`).

Następnie tata wyniósł rzeczy do garażu sąsiada. W tym momencie przedmioty zostały więc przeniesione z repozytorium lokalnego do zdalnego. Odpowiednia komenda to `git push`.

Wyprzedaże organizowane są po to, by rzeczy, które nam już nie są potrzebne, mogły zostać zakupione przez innych. My je „wypychamy" (`push`), dobrze byłoby jednak, by inni mogli je pobrać (`pull`). Aby to było możliwe, dla wybranych przedmiotów należy

wykonać polecenie `git pull`. Nie będziemy jednak przeprowadzać transakcji w garażu. Wybrane przez kupującego rzeczy ułożymy więc znów w naszym repozytorium lokalnym — w salonie (rysunek 1.10).



Rysunek 1.10. Ilustracja przepływu zadań (ang. workflow) w Gicie na przykładzie wyprzedaży garażowej

Podobnie będzie wyglądała praca w Gicie z kodem. Dopóki pliki są u nas na dysku, to są lokalne (ang. *local*); gdy już przerzucimy je do repozytorium, to oprócz plików lokalnych mamy też ich zdalne (ang. *remote*) kopie.

Przykład 1.2

Wróćmy do problemu z kodem źródłowym warzywniaka. Za pomocą systemu kontroli wersji można go rozwiązać w opisany tutaj sposób. Na początku kolega pracujący nad integracją sklepu z systemem płatności PayPal mógł zrobić kopię kodu z dnia 10.10.2019 i pracować właśnie na niej. Tak zrobioną kopię nazywamy gałęzią. Komenda, która służy do tworzenia nowej gałęzi, to `git checkout -b paypal` (`checkout` utworzy lokalnie gałąź o nazwie `paypal`; nazwa będzie zatem odpowiadać rozwijanej funkcji). Na końcu prac trzeba scalić wszystkie aktywne gałęzie z oryginalnym kodem źródłowym, przechowywanym w głównej (ang. *master*) gałęzi. Uzyskuje się to za pomocą komendy `git merge paypal` wykonywanej, gdy jesteśmy w gałęzi *master*.

Prace nad projektem rozpoczynamy od utworzenia projektu w dowolnym portalu, który obsługuje zarządzanie wersjami projektu (np. *github.com*, *bitbucket.org* — my będziemy pracowali w pierwszym z nich). Po założeniu konta w serwisie *github.com* (czyli serwerze z kodami źródłowymi) i utworzeniu projektu o nazwie *firstProject* uzyskamy możliwość

pobrania projektu poprzez bezpośredni link, który otrzymaliśmy. Będzie on wyglądać podobnie do tego: *https://github.com/nazwauzytkownika/firstProject.git*.

URL jest dostępny od razu po utworzeniu projektu, dzięki czemu możemy go udostępniać współpracownikom.

1.2.1.2. WebStorm — IDE

W repozytorium zdalnym Gita będziemy przechowywać nasz kod. Kod możemy pisać w dowolnym programie (może to być nawet notatnik), ale zrobimy to o wiele sprawniej, jeśli użyjemy IDE (akronim od ang. *Integrated Development Environment* — zintegrowane środowisko programistyczne), przede wszystkim ze względu na oferowane przezeń funkcje, jak uzupełnianie składni czy kontrola błędów. Możemy użyć np. programu WebStorm, który można pobrać ze strony producenta (wybieramy plik *.exe*):

https://www.jetbrains.com/webstorm/download/download-thanks.html?platform=windows

Dobrze jest zapoznać się z przykładowym uruchomieniem pierwszego projektu; opis tego procesu można znaleźć na stronie producenta:

https://www.jetbrains.com/help/webstorm/getting-started-with-webstorm.html

UWAGA

Jakie środowisko wybrać? Trudno odpowiedzieć na to pytanie. Wybór jest w tym przypadku (jak i w wielu innych) arbitralny, zależy głównie od indywidualnych preferencji użytkownika. Trzeba jednak zdawać sobie sprawę, że funkcjonalność środowisk programistycznych jest bardzo podobna (podstawowe funkcje to: kolorowanie kodu, autouzupełnianie poleceń, sygnalizowanie błędów składniowych, tryb debugowania kodu). Jeżeli opanujesz jedno, prawdopodobnie poradzisz sobie z pozostałymi. Zazwyczaj różnią się od siebie wyglądem, skrótami klawiszowymi i liczbą zintegrowanych narzędzi.

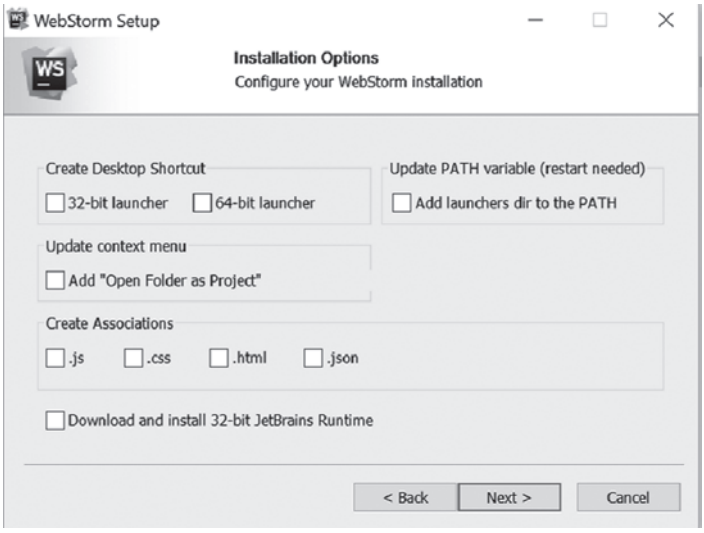
Przykłady prezentowane w tym podręczniku wykorzystują wspomniany już WebStorm. Jest to rozwiązanie komercyjne, jego autorzy jednak udostępniają darmową licencję do celów edukacyjnych lub też oferują zniżki dla start-upów. Do wyboru tego narzędzia skłania m.in. jego dobra integracja z narzędziem Jira.

Można jednak skorzystać z innych środowisk. Wiele z nich jest — jak WebStorm — komercyjnych, ich producenci udostępniają jednak darmowe wersje z nieco ograniczoną funkcjonalnością (przy czym dla początkującego programisty będzie ona zupełnie wystarczająca). Tak jest choćby w przypadku IntelliJ IDEA (*https://www.jetbrains.com/idea/*). Obok płatnej wersji Ultimate funkcjonuje darmowa, bez ograniczeń czasowych, wersja Community. Niestety ta ostatnia nie obsługuje języka JavaScript, dostępnego w wersji płatnej, jeśli jednak chcesz programować w innych językach (przede wszystkim w Javie), warto ją wypróbować. Visual Studio, program Microsoftu obsługujący m.in. JavaScript, przygotowano w aż trzech wersjach: komercyjnych Professional i Enterprise (obie posiadają bezpłatne wersje próbne) oraz darmowej Community. Wszystkie można pobrać ze strony *https://visualstudio.microsoft.com/pl/*.

UWAGA cd.

Istnieją także środowiska zupełnie darmowe, jak choćby te dostępne na licencji *open source* (tzn. że aplikacja jest tworzona przez programistów w ramach otwartego projektu i każdy może pobrać kod oraz zmodyfikować go na własny użytek, może też dopisać nowe funkcje i udostępnić je innym), takie jak Visual Studio Code (najczęściej używany edytor kodu w przypadku języków opartych na JavaScripcie, dostępny pod adresem *https://github.com/microsoft/vscode*), Netbeans (*https://netbeans.org/index_pl.html*) czy Eclipse (*https://www.eclipse.org/*). Wartym polecenia środowiskiem jest także Atom (*https://atom.io/*).

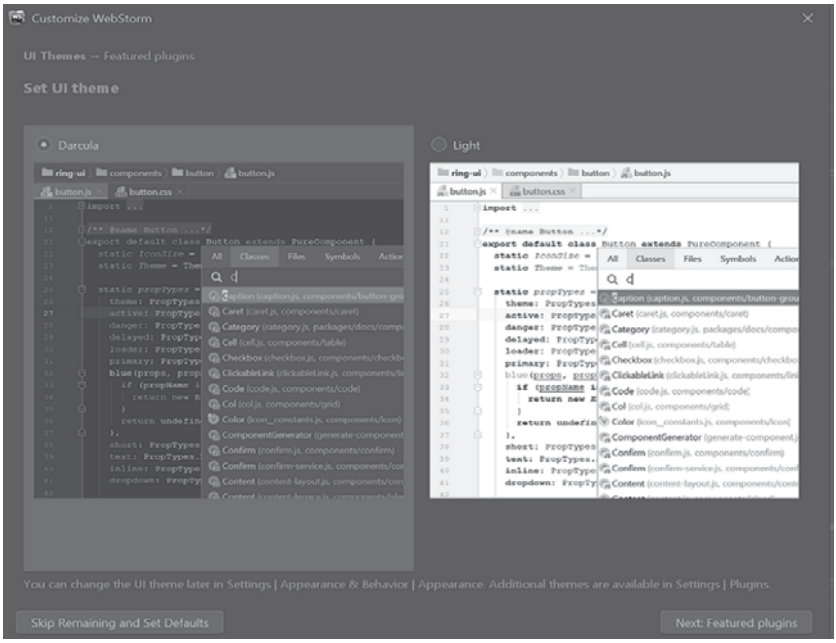
Podczas instalacji warto zwrócić uwagę na okno dotyczące opcji w sekcji *Create Associations* (rysunek 1.11) . Mamy do wyboru rozszerzenia plików, które mają być skojarzone z programem — to znaczy, że będą domyślnie otwierane przez WebStorm. Zalecamy wybranie plików z rozszerzeniami *.js* oraz *.html*.



Rysunek 1.11. Wybór opcji dostępnych podczas instalacji programu WebStorm

Przy pierwszym uruchomieniu pojawi się okno wyboru stylu programu. Spora grupa programistów ze względu na minimalizowanie odbicia światła wybiera ciemny styl *Darcula* (rysunek 1.12), ale są dostępne także inne style. Ich wybór jest zależny od indywidualnych preferencji użytkownika.

Nie trzeba instalować dodatkowych wtyczek, można więc pominąć kolejne kroki i pozostawić domyślne ustawienia (w razie potrzeby wtyczki mogą być doinstalowane później).



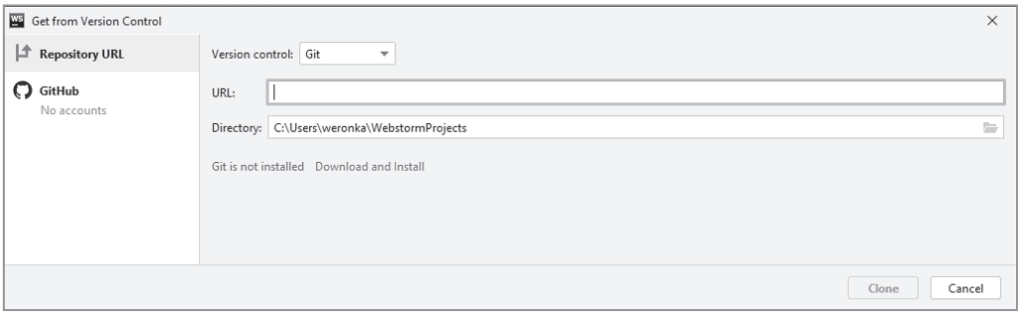
Rysunek 1.12. Wybór stylu dla programu WebStorm

Po zainstalowaniu oprogramowania pojawi się okno (rysunek 1.13) pozwalające wybrać, czy chcemy utworzyć nowy projekt (*Create New Project*), czy otworzyć i edytować już istniejący (*Open*). Opcja *Get from Version Control* pozwala na pobranie kodu z repozytorium zdalnego.

Rysunek 1.13. Okno, w którym wskazujemy źródło kodu

1.2.1.3. Integracja IDE z Gitem

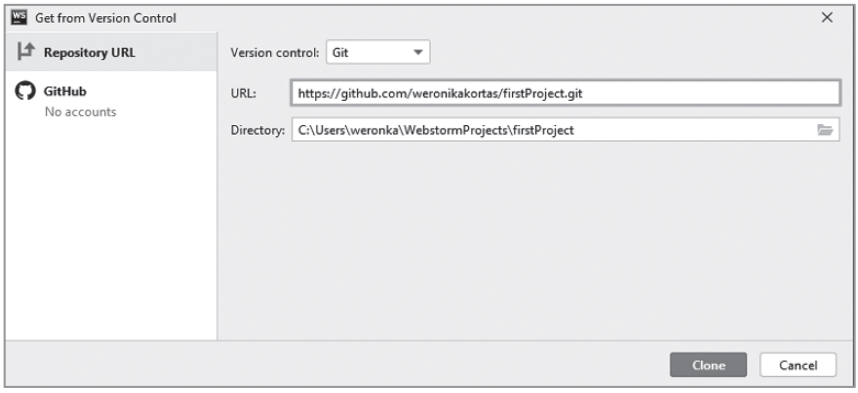
Przy pierwszym uruchomieniu programu WebStorm i wybraniu opcji pobrania kodu z repozytorium uzyskamy widok podobny do pokazanego na rysunku 1.14:



Rysunek 1.14. Okno dialogowe ze wskazaniem łącza do repozytorium

Oczywiście istnieją inne repozytoria projektowe, ale na początku warto się skupić na najbardziej popularnym narzędziu.

Jeśli do tej pory nie zainstalowaliśmy Gita, to wyświetlony zostanie link pozwalający na doinstalowanie brakujących aplikacji (*Download and Install* na rysunku 1.14). Po poprawnym zainstalowaniu pojawi się okno podobne do tego z rysunku 1.15:



Rysunek 1.15. Okno dialogowe pozwalające na pobranie pustego projektu z repozytorium zdalnego

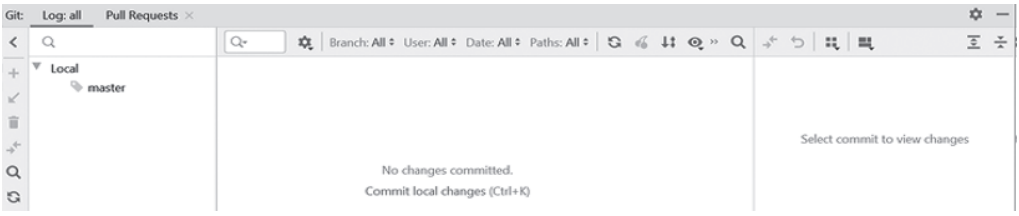
Adres URL określa, w jakiej zdalnej lokalizacji mamy nasz projekt (czyli w polu URL trzeba wkleić link do projektu utworzonego na repozytorium zdalnym). Dynamicznie dodał się również katalog na podstawie wybranego projektu. Po kliknięciu przycisku *Clone* w tle będzie się wykonywać komenda:

```
git clone https://github.com/nazwa_uzytkownika/firstProject.git
```

Po otwarciu pustego projektu w WebStorm można zobaczyć w lewym dolnym rogu opcję wyboru repozytorium Gita (rysunek 1.16). Możemy ją wywołać za pomocą skrótu klawiszowego *Alt+9*.

Rysunek 1.16. Lewy dolny róg z opcją wyboru okna dialogowego prowadzącego do repozytorium Gita

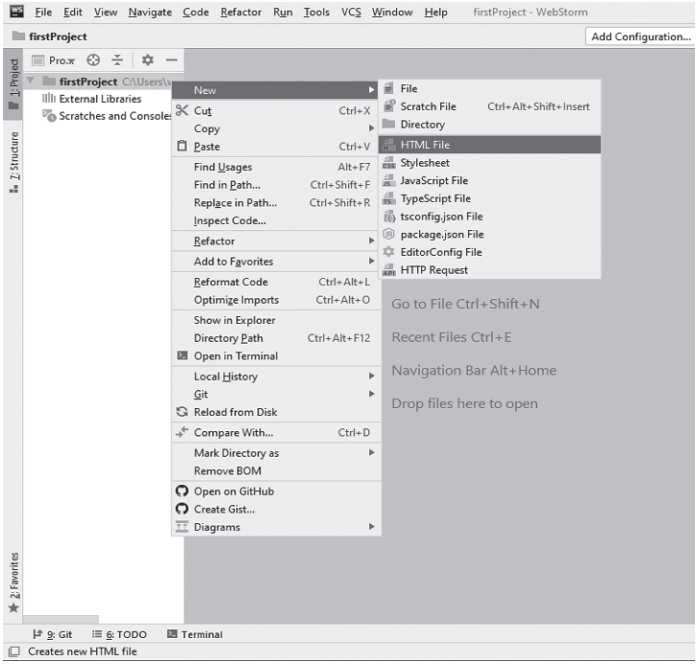
Kiedy uruchomimy to okno, zobaczymy, że obecnie nie mamy żadnych zmian (czyli nic nie stworzyliśmy — *No changes committed*), ale w lewej części okna widać, że jesteśmy na lokalnej gałęzi *master* (rysunek 1.17).



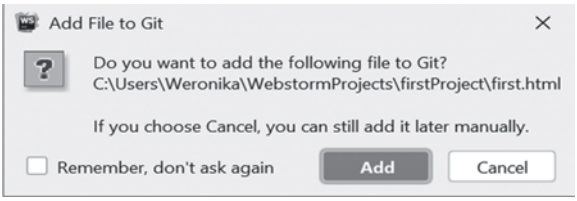
Rysunek 1.17. Widok okna Git w WebStorm

Teraz dodamy nowy plik, aby został umieszczony w repozytorium. W tym celu kliknij na nazwie projektu prawym przyciskiem myszy i wybierz opcje zgodnie z rysunkiem 1.18. Po uzupełnieniu nazwy pliku i wybraniu dowolnej wersji HTML pojawi się okno dialogowe (rysunek 1.19).

Rysunek 1.18. Widok dodania nowego pliku do istniejącego projektu

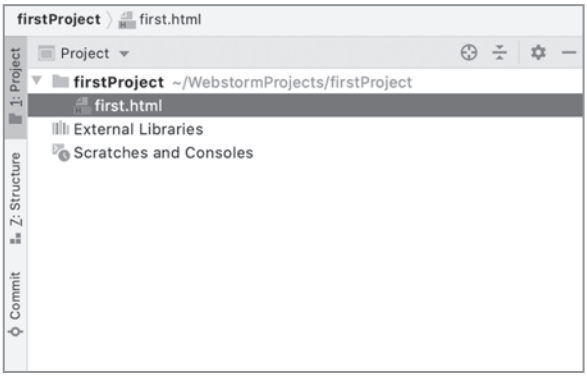


Jeśli się zgodzimy, to zobaczymy nasz plik w oknie lokalnych zmian (*Local Changes — Commit*).



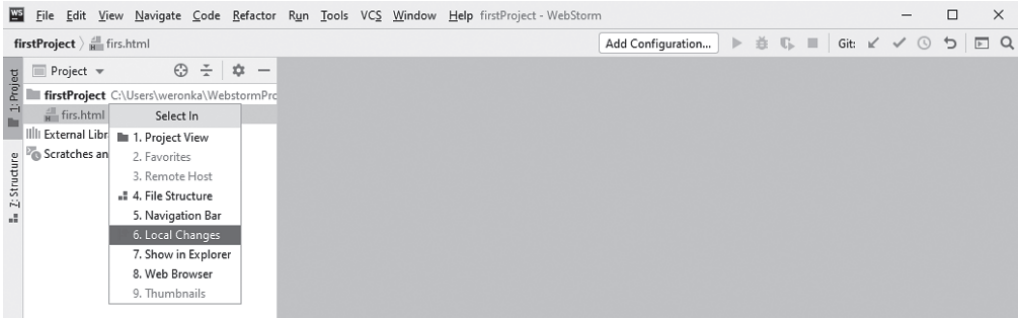
Rysunek 1.19. Okno dialogowe z zapytaniem, czy nowo powstały plik dodać do repozytorium Gita

Jeśli domyślnie się nie otworzyło, mamy możliwość jego wyboru (zakładka *Commit* z lewej strony na rysunku 1.20; ten sam efekt uzyskamy, wciskając skrót *Alt+F1*).



Rysunek 1.20. Wybór dostępnych widoków w WebStorm (zakładki z lewej strony)

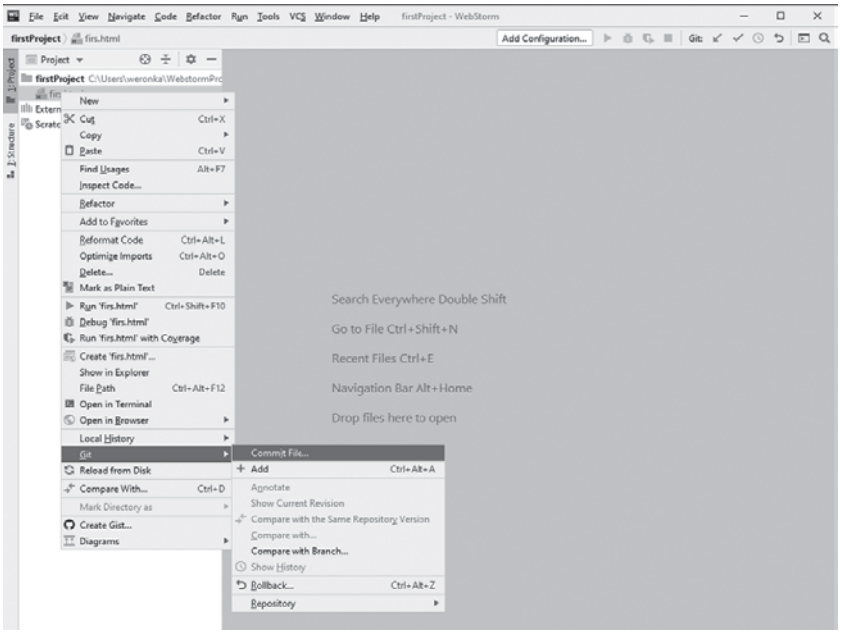
Po wciśnięciu wspomnianej zakładki (lub skrótu) uzyskamy możliwość wyboru widoku (rysunek 1.21). Pod opcją z numerem 6 znajdziemy skrót prowadzący do lokalnych zmian (*Local Changes*).



Rysunek 1.21. Okno dialogowe wyboru dostępnych widoków

Po wybraniu tej opcji zobaczymy, że na liście lokalnych zmian nazwa dodanego pliku jest wypisana kolorem zielonym (gdybyśmy w oknie pokazanym na rysunku 1.19 wybrali przycisk *Cancel*, wyświetlałaby się na czerwono).

W oknie lokalnych zmian po wybraniu zmian możemy z menu kontekstowego wybrać opcję *Commit File*, tym samym jawnie określając, że te zmiany chcemy przenieść do repozytorium zdalnego (rysunek 1.22).



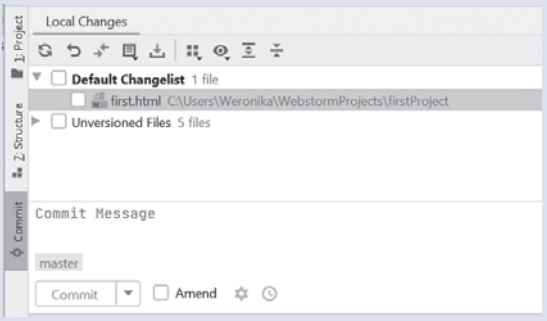
Rysunek 1.22. Widok menu kontekstowego z oznaczonych wyborem Commit File

W oknie dialogowym, które się teraz ukaże, możemy jeszcze odznaczyć pliki wcześniej zaznaczone lub też wybrać pliki nieoznaczone, znajdujące się w katalogu *Unversioned Files*. Wymagane jest wprowadzenie opisu zatwierdzanych zmian (*Commit Message*).

UWAGA

Zmiany zatwierdzane w repozytorium zdalnym nazywane są często **commitem** (rysunek 1.23). Tak też będą określane w dalszej części tego rozdziału.

Rysunek 1.23. Okno dialogowe Commit



UWAGA

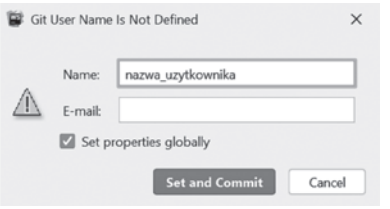
Im dokładniejsze są opisy zatwierdzanych zmian, tym łatwiej jest później nimi zarządzać.

W menu kontekstowym commita (rysunek 1.24) (po uzupełnieniu wymaganych pól — czyli wybraniu pliku/plików oraz uzupełnieniu pola z komentarzem) mamy do wyboru tylko przeniesienie zmian do tymczasowego katalogu (*Commit*). W rzeczywistości wszystkie pliki są oznaczone jako do przeniesienia.

Rysunek 1.24. Menu kontekstowe Commit

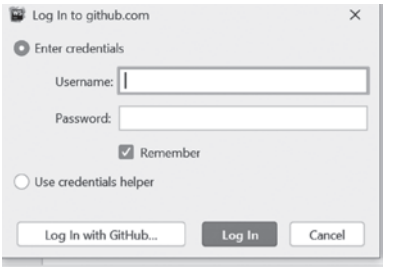
Kolejnym krokiem jest wysłanie zmian do repozytorium zdalnego. Za pierwszym razem pojawi się okno powiadamiające o tym, że nie mamy ustawionych niezbędnych informacji wskazujących, kto wykonał zmiany oraz jaki jest adres e-mail użytkownika (rysunek 1.25).

Rysunek 1.25. Okno dialogowe do uzupełnienia danych wymaganych przy komunikacji z repozytorium zdalnym

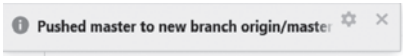


Jeśli wybraliśmy opcję z *Commit and Push*, to od razu zostaliśmy przekierowani do okna dialogowego z prośbą o podanie informacji niezbędnych do uwierzytelnienia w repozytorium zdalnym (rysunek 1.26).

Rysunek 1.26. Okno dialogowe z prośbą o dane dostępowe do wskazanego wcześniej repozytorium zdalnego

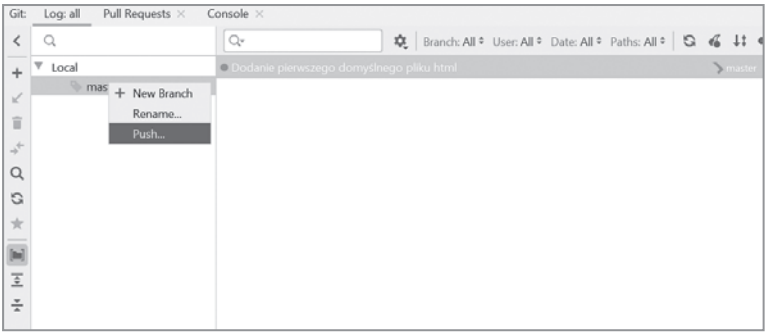


Jeśli poprawnie wpisaliśmy dane i operacja zakończyła się sukcesem, to w prawym dolnym rogu programu WebStorm pojawi się komunikat (rysunek 1.27):



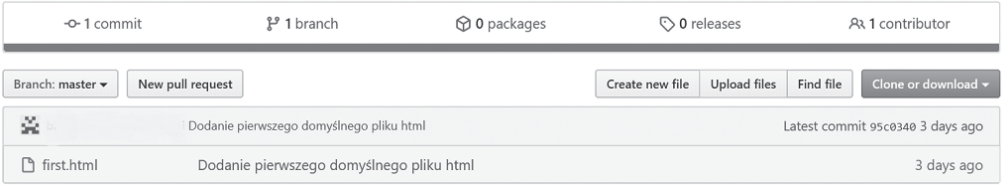
Rysunek 1.27. Komunikat o poprawnym wysłaniu commita do repozytorium zdalnego

Jeśli nie wybraliśmy *Commit and Push*, to po wybraniu widoku *Git* (dostępnego również pod skrótem klawiaturowym *Alt+9*) i po wskazaniu gałęzi *master* w menu kontekstowym można wybrać *Push* i postępować zgodnie z wcześniej opisanymi krokami (rysunek 1.28).



Rysunek 1.28. Menu kontekstowe w widoku Git

Po poprawnym przejściu całego procesu pod wskazanym linkiem projektu pojawi się wpis na stronie wybranego przez nas repozytorium zdalnego, wskazujący na to, że commit został zakończony sukcesem (rysunek 1.29).

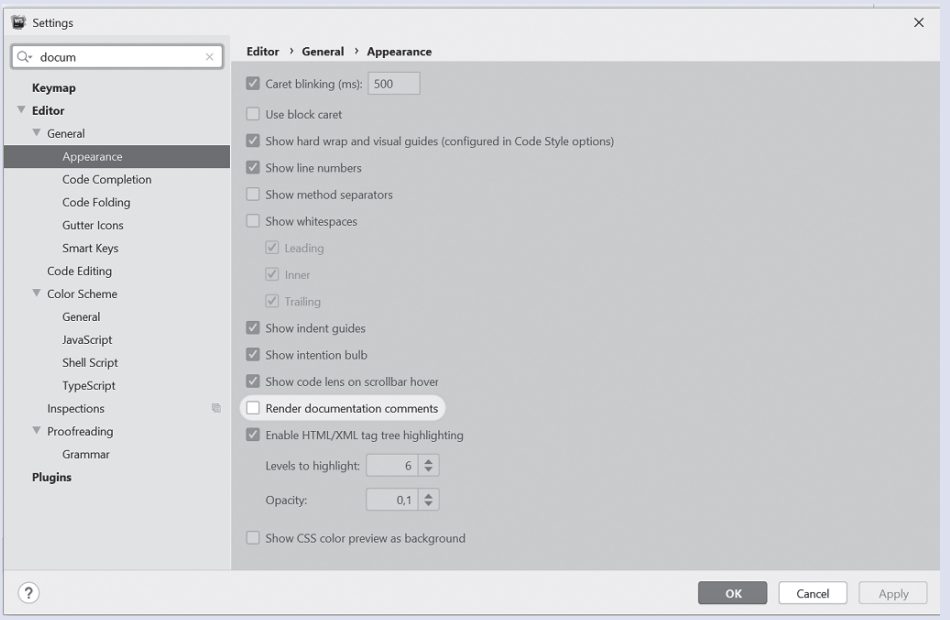


Rysunek 1.29. Widok na stronie repozytorium zdalnego z pierwszym commitem

UWAGA

Podczas pisania aplikacji możemy nakazać środowisku WebStorm, by automatycznie generowało dokumentację.

Wywołamy tę opcję poprzez przejście do ustawień (*File/Settings*) albo poprzez skrót klawiszowy *Ctrl+Alt+S* (rysunek 1.30).



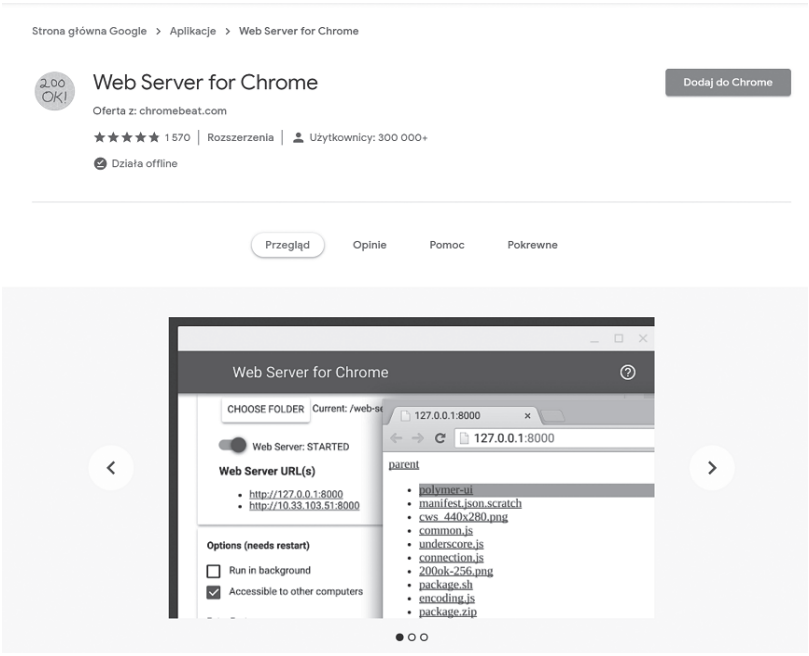
Rysunek 1.30. Wybierając wyróżnione pole, nakazujemy IDE generowanie dokumentacji

1.2.2. Serwer aplikacji

Oprócz plików HTML i CSS większość dostępnych stron internetowych ma również fragmenty pozwalające na interakcję z użytkownikiem, napisane w języku JavaScript. W celu poprawnego uruchomienia wszystkich plików składających się na aplikację należy je udostępnić na serwerze i wskazać adres URL, pod którym jest dostępny nasz program.

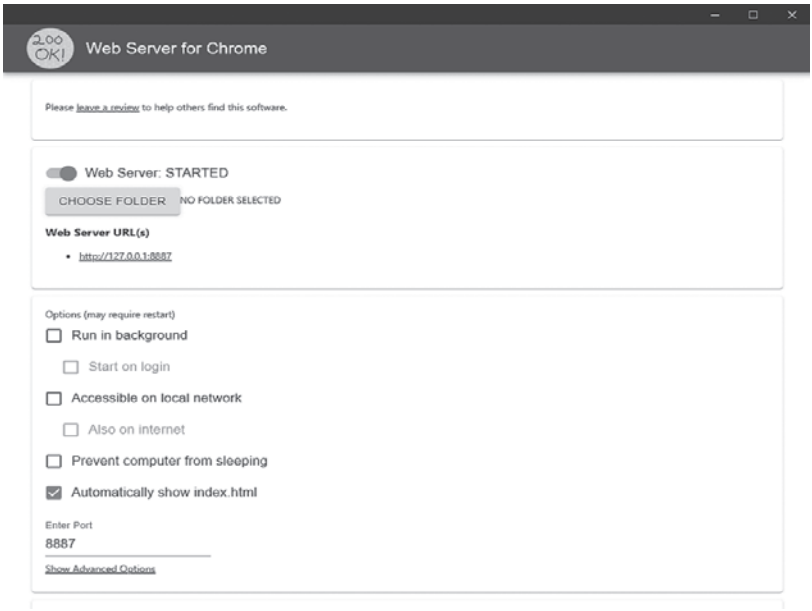
Przeglądarka Chrome ma wtyczkę, która jest właśnie serwerem.

Aby korzystać z tej funkcji, należy wyszukać i zainstalować wtyczkę *Web Server for Chrome* (rysunek 1.31).



Rysunek 1.31. Pobranie wtyczki Web Server for Chrome

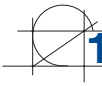
W celu uruchomienia serwera należy wybrać w sekcji *Rozszerzenia* wtyczkę *Web Server for Chrome*; pojawi się wówczas okno dialogowe z ustawieniami wtyczki (rysunek 1.32):



Rysunek 1.32. Okno ustawień wtyczki Web Server for Chrome

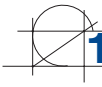
W opcji *Choose folder* należy wskazać folder, gdzie jest zlokalizowana aplikacja (czyli folder nadrzędny dla pliku *index.html*).

Po kliknięciu URL ujrzymy uruchomioną aplikację.



1.3. Podsumowanie

Rozpoczynając pracę w zespole projektowym, na instalację narzędzi oraz zapoznanie się z nimi mamy od kilku godzin do kilku dni, w zależności od liczby narzędzi i ilości czasu, jaki został oszacowany przez przełożonego. Zapoznaj się dobrze z opisanymi tu programami, bo w dalszej części książki wszystkie będą wielokrotnie używane. Oprócz przedstawionych tutaj wybranych opcji warto przyswoić sobie dokumentację udostępnioną dla wyżej wymienionych aplikacji. Czas poświęcony na poznawanie narzędzi zwróci się w dwójnasób podczas korzystania z nich.



1.4. Zadania

Zadanie 1.1

Jaka jest różnica pomiędzy walidacją a weryfikacją?

Zadanie 1.2

Czy GitHub to jedyna platforma do repozytorium zdalnego? Jeśli uważasz, że nie, wymień inne.

Skorowidz

J

K

L

M

N

O

P

R

S

Ś

T

U

W

Y

Z

Część 1
Inżynieria programowania – projektowanie oprogramowania, testowanie i dokumentowanie aplikacji

Kwalifikacja INF.04

Projektowanie, programowanie i testowanie aplikacji

Podręcznik do nauki zawodu
technik programista

Informatycy i programiści należą obecnie do najbardziej poszukiwanych specjalistów. Stąd tytuł, który uzyskuje się po szkole średniej, nie stanowi jedynie świadectwa ukończenia pewnego etapu edukacji. **Technik programista** to zawód o wymiernej wartości rynkowej. Absolwenci tego kierunku kształcenia nie mają większych problemów ze znalezieniem intratnego i rozwojowego zajęcia, a pracodawcy chętnie inwestują w ich szkolenia, by mogli zdobywać coraz wyższe kwalifikacje. Wśród bazowych umiejętności, które powinien posiadać specjalista w tej dziedzinie, są podstawy programowania: projektowanie kodu, testowanie go, dokumentowanie swojej pracy oraz analizowanie pracy innych programistów.

Treści, które zawiera *Kwalifikacja INF.04. Projektowanie, programowanie i testowanie aplikacji. Część 1. Inżynieria programowania – projektowanie oprogramowania, testowanie i dokumentowanie aplikacji. Podręcznik do nauki zawodu technik programista*, opierają się na podstawie programowej kształcenia w zawodzie technika programisty. Są powiązane merytorycznie, strukturalnie i metodycznie z ogólnymi celami kształcenia określonymi w kwalifikacji INF.04. Projektowanie, programowanie i testowanie aplikacji.

Publikację podzielono na siedem rozdziałów, które szczegółowo omawiają poruszone w nich zagadnienia:

- Przygotowanie środowiska pracy
- Elementy oprogramowania na podstawie języka JavaScript
- Projektowanie aplikacji
- Architektoniczne projektowanie aplikacji
- Testowanie oprogramowania
- Tworzenie dokumentacji testowej
- Metodologie prowadzenia projektu informatycznego

Podręcznik do nauki zawodu technik programista to charakteryzujący się wysoką jakością kompletny zestaw edukacyjny przygotowany przez dysponującego ogromnym doświadczeniem lidera na rynku książek informatycznych — wydawnictwo Helion.

W skład zestawu podręczników do kwalifikacji INF.04 wchodzą także:

- *Kwalifikacja INF.04. Projektowanie, programowanie i testowanie aplikacji. Część 2. Programowanie obiektowe. Podręcznik do nauki zawodu technik programista*
- *Kwalifikacja INF.04. Projektowanie, programowanie i testowanie aplikacji. Część 3. Aplikacje internetowe. Podręcznik do nauki zawodu technik programista*
- *Kwalifikacja INF.04. Projektowanie, programowanie i testowanie aplikacji. Część 4. Aplikacje mobilne. Podręcznik do nauki zawodu technik programista*
- *Kwalifikacja INF.04. Projektowanie, programowanie i testowanie aplikacji. Część 5. Aplikacje desktopowe. Podręcznik do nauki zawodu technik programista*

Podręczniki należące do tej serii przygotowano z myślą o wykształceniu kompetentnych techników, którzy bez trudu poradzą sobie z wyzwaniami, jakie stawia przed nimi współczesna informatyka. Wiedza zawarta w serii pomoże zdać egzamin zawodowy i uzyskać umiejętności praktyczne, przydatne w przyszłej pracy.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

ISBN 978-83-283-7415-7
9 788328 374157

INFORMATYKA W NAJLEPSZYM WYDANIU